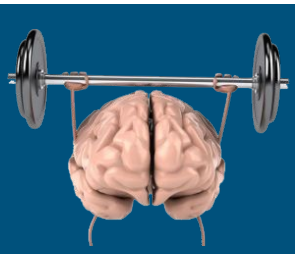
Aula 06

Programação Orientada a Objetos

emerson@paduan.pro.br

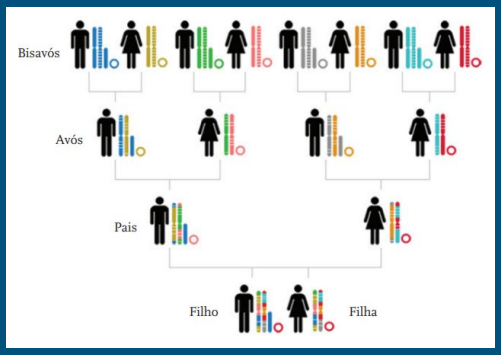
Exercício 6-0



Preparando nossa pasta de hoje com Git / GitHub

emerson@paduan.pro.br

Herança



receber dos antepassados

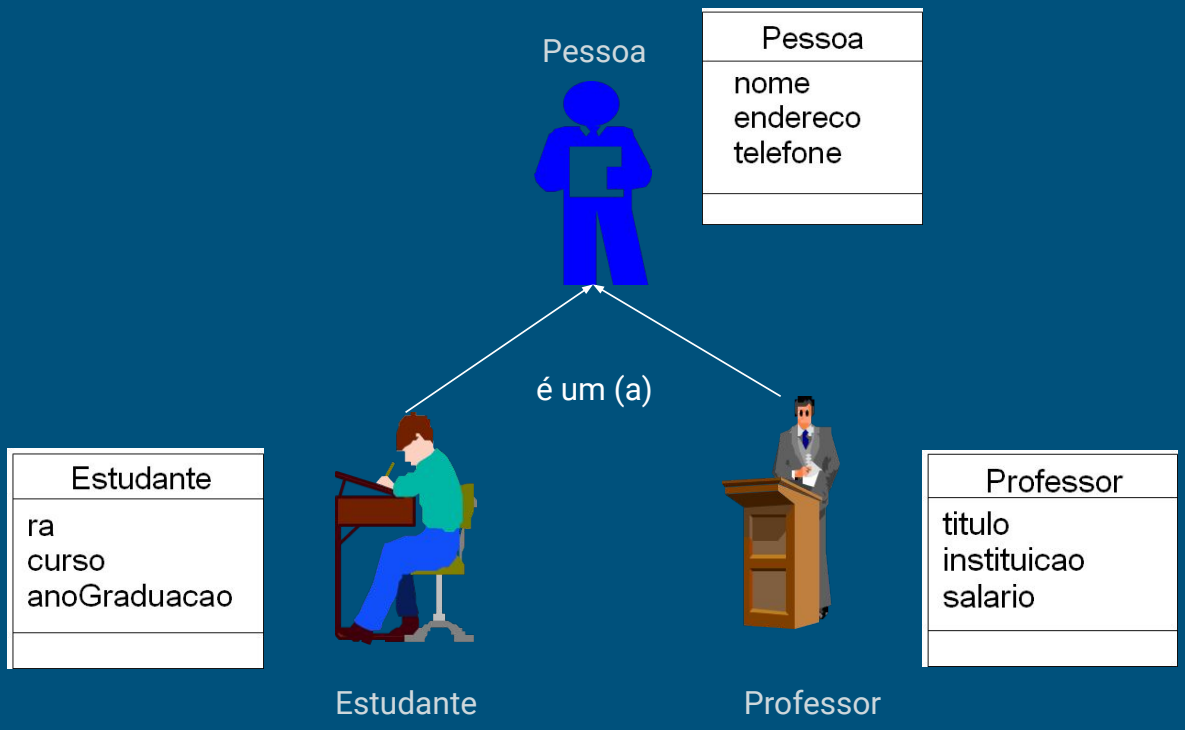
emerson@paduan.pro.br

Herança

Herança é um mecanismo que permite que características comuns a diversas classes sejam derivadas de uma classe base, ou superclasse.

A herança é uma forma de reutilização de software em que novas classes são criadas a partir das classes existentes, herdando seus atributos e métodos e adicionando novos recursos que as novas classes exigem.

emerson@paduan.pro.br



Em Java

```
public class SuperClass {
    //corpo da superclasse...
}
```

extends - indica que a criação de uma nova classe que **herda** de uma classe existente

```
public class SubClass extends SuperClass{
    //corpo da subclasse...
}
```

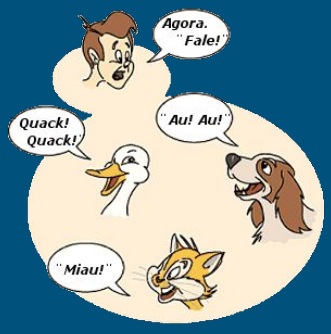
super

Palavra-chave *super* refere-se a uma superclasse.

Pode indica a chamada ao construtor da superclasse ou ser utilizada para invocar métodos da superclasse dentro da subclasse.

emerson@paduan.pro.br

Polimorfismo



Há muitas formas de "falar"

emerson@paduan.pro.br

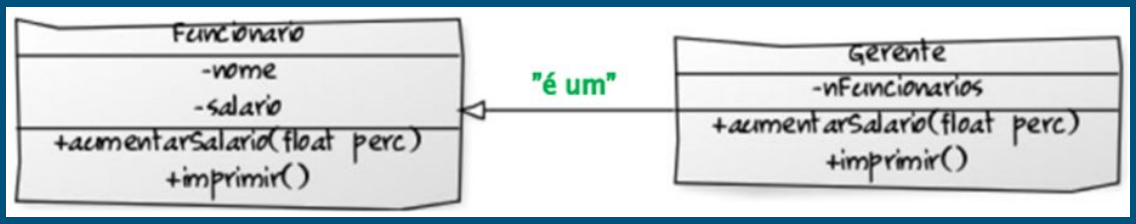
Polimorfismo

Um método definido em uma *subclasse* com o mesmo nome e mesma lista de parâmetros que um método em uma de suas classes antecessoras **oculta** o método da classe ancestral a partir da subclasse.

Let's Code

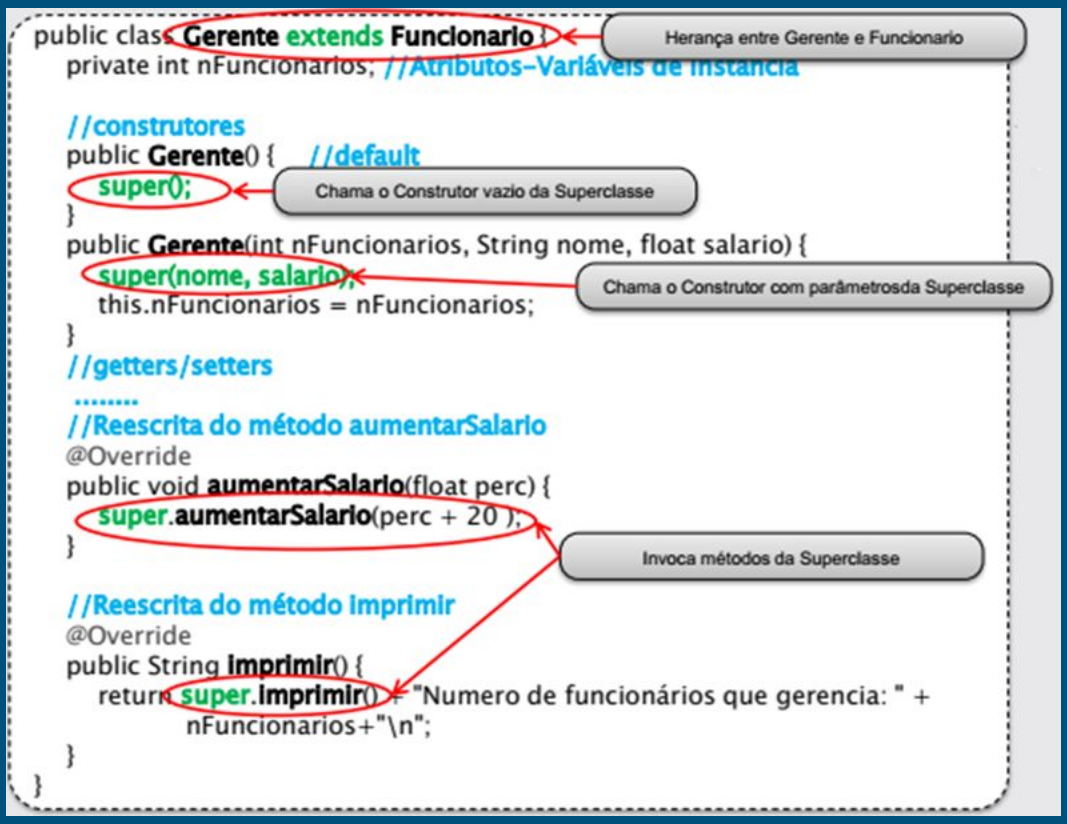
Vamos criar classes para testar esses conceitos.

Exemplo



Sabe-se que o gerente recebe um bônus adicional de 20% quando um aumento de salário é efetivado. Os demais funcionários recebem o aumento de acordo com o percentual informado.


```
public class Funcionario {
   //Atributos – Variáveis de Instância
   private String nome;
   private float salario;

   //Construtor
   public Funcionario() { } //default
   //sobrecarregado
   public Funcionario(String nome, float salario) {
      this.nome = nome;
      this.salario = salario;
   }

   //getters/setters
   .....

   //Métodos da classe
   public void aumentarSalario(float perc){
      this.salario += this.salario * perc/100.0;
   }

   public String imprimir(){
      return "Funcionario: " + nome +
            "\nSalário: R$ " + String.format("%.2f \n", salario);
   }
}
```



Observe

Reescrita do método aumentarSalario()

– Para funcionar diferente para gerentes e funcionários comuns (gerente recebe um bônus adicional de 20%).

– Esse método não tem acesso direto às variáveis de instância privados da superclasse, ou seja, esse método não pode alterar diretamente a variável de instância *salario*, embora cada objeto Gerente tenha uma variável de instância *salario*.

– Modificador de acesso ***protected*** – dá acesso direto aos atributos da superclasse pela subclasse!

```
public class AppFuncionarioGerente {
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);

        Funcionario func = new Funcionario("Jose da Silva", 1000.0f);
        Gerente ger = new Gerente(45, "Joao Medeiros", 5000.0f);

        //calcular 10% de aumento de salário para os funcionarios
        func.aumentarSalario(10);
        ger.aumentarSalario(10);

        System.out.println("===== DADOS DO FUNCIONÁRIO =====");
        System.out.println(func.imprimir());
        System.out.println("===== DADOS DO FUNCIONÁRIO =====");
        System.out.println(ger.imprimir());

    }
}
```


Exercício 6-1

Construir uma classe para representar um funcionário com os seguintes atributos: nome, horas trabalhadas e valor pago por hora trabalhada. Implementar um método para calcular e retornar o salário final de um funcionário e um método imprimir.

Criar uma subclasse para representar um funcionário sênior. A diferença entre eles é que um funcionário sênior recebe um bônus a cada 10 horas trabalhadas. O bônus é um atributo do funcionário sênior. Sobrescrever os métodos calcularSalario() e imprimir().

Criar uma classe AppFuncionario para instanciar objetos da classe Funcionario e Senior e realizar chamada a seus métodos.

Upcast

É uma conversão na qual subclasses são promovidas a superclasses.

Característica: A conversão é implícita! A promoção é realizada automaticamente!

Exemplo: Gerente "é um" Funcionario

```
Funcionario func = new Gerente(); //upcast
```


Downcast

É a operação inversa, superclasses são convertidas em subclasses.
Característica: A conversão é explícita! Tem que indicar o TIPO!

Exemplo:

```
Funcionario func = new Gerente(); //upcast

Gerente ger = (Gerente) func; //downcast
```

Coleções

ArrayList

ArrayList<T> (pacote java.util) pode alterar dinamicamente seu tamanho para acomodar mais elementos.

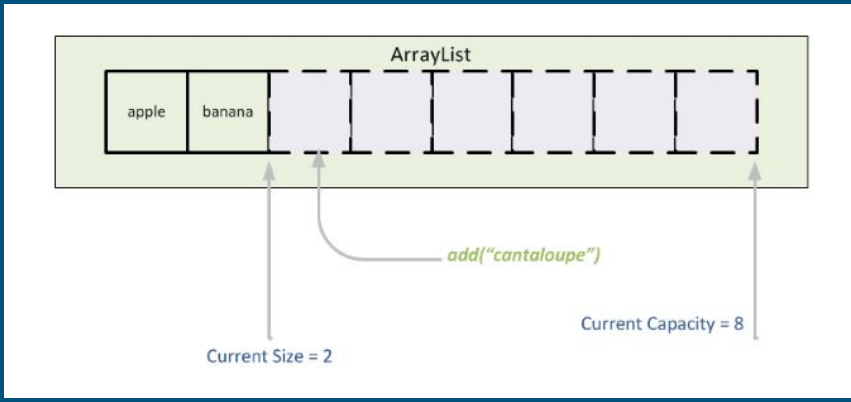
– *T* indica o tipo de elemento armazenado na coleção

– Isso é semelhante a especificar o tipo ao declarar um array, exceto que apenas tipos *não-primitivos* podem ser utilizados com essas classes de coleção.

Classes com essa espécie de marcador de lugar são chamadas *classes genéricas*.

ArrayList

Coleção em Java que permite armazenar elementos de forma DINÂMICA.



ArrayList

Exemplo:

```
import java.util.ArrayList;

public class Exemplo {
	public static void main(String args[])
	{
		ArrayList<String> nomes = new ArrayList<>();

		nomes.add("Huguinho");
		nomes.add("Zezinho");
		nomes.add("Luizinho");

		System.out.println(nomes);
	}
}
```

Principais métodos

Método	Descrição
add(Object o)	Adiciona um elemento ao fim do ArrayList
add(int index, Object o)	Adiciona um elemento no índice especificado do ArrayList
clear()	Remove todos os elementos do ArrayList
get(int index)	Retorna o elemento do índice especificado
indexOf(Object o)	Retorna o índice da primeira ocorrência do elemento especificado no ArrayList
remove(Object o)	Remove a primeira ocorrência do valor especificado
remove(int index)	Remove o elemento do índice especificado
size()	Retorna o número de elementos armazenados no ArrayList
isEmpty()	Retorna true se não existem elementos no ArrayList

emerson@paduan.pro.br

foreach

foreach – (para cada)

iterar sobre coleções de maneira simples e direta

Sintaxe:

```
for( tipo variavel : nomeArray ){
    //corpo do for
}
```

```
int vetor[] = {1,2,3,4,5,6,7,8,9,10};
int  somatorio = 0;

for( int valor : vetor) {
        somatorio += valor;
}
System.out.println(somatorio);
```

emerson@paduan.pro.br

Exemplo

Utilizando as classes do exercício anterior, criar a Classe para Gerenciar funcionário que contenha como atributo um ArrayList de Funcionários.

Implementar os seguintes métodos:

– adicionar() : para armazenar os funcionários no ArrayList;

– listar() : retorna uma String com os dados de todos os funcionários;

– listarSalarios(): retorna uma String contendo o nome e o salário de todos os funcionários cadastrados no ArrayList;

Exercício 6-2

Acrescente os seguintes métodos:

– totalDeFuncionariosSenior() : retorna um inteiro indicando o total de funcionários senior cadastrados no ArrayList. Para diferenciar os objetos utilizar *instanceOf*

– funcionarioComMaiorBonus(): retorna o nome e o bonus calculado do funcionário Senior que recebeu o maior bônus entre todos os cadastrados no ArrayList;

– listarPorFaixaSalarial() : método que recebe uma faixa salarial inicial e final e retorna o nome e o salário dos funcionários dentro da faixa informada;

Finalizando

Criar uma aplicação com o seguinte menu de opções:

```
1 - Cadastrar Funcionário
2 - Listar
3 - Listar salarios
4 - Total de Funcionários Sênior
5 - Funcionário com Maior Bônus
6 - Listar por Faixa Salarial
7 - Sair
```


Exercício 6-3

Como remover funcionários ???

Associação entre classes

emerson@paduan.pro.br

Associação

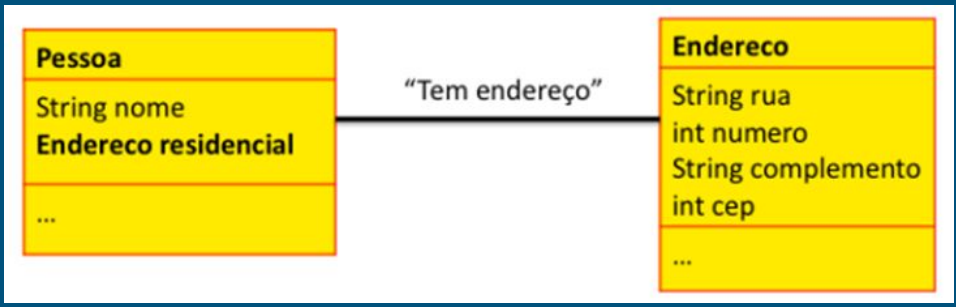
Um sistema é composto por várias Classes.

• As classes se conectam para poderem se comunicar por troca de mensagens (chamadas de métodos)

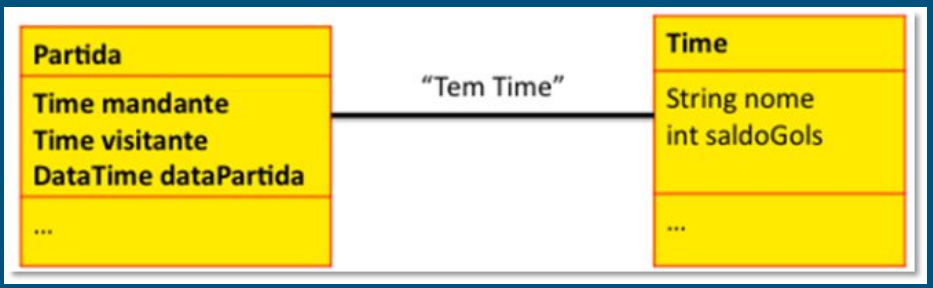
• Quando um ou mais atributos de uma Classe é uma *referência* para outra Classe temos uma **associação**.

emerson@paduan.pro.br

Exemplo



Exemplo 1: N



Em Java

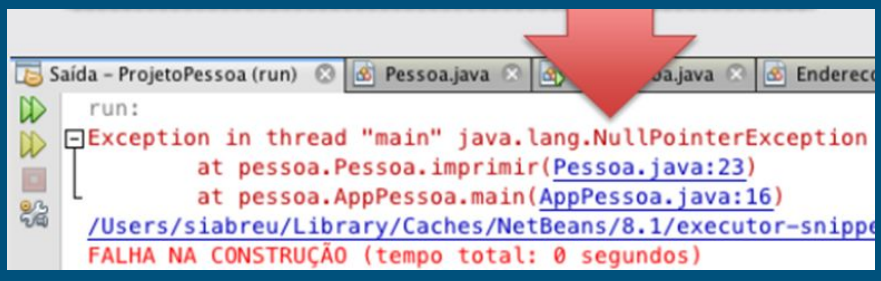
```
public class Pessoa{
    //atributos
    private String nome;
    private int idade;
    private char sexo;
    private Endereco end;
    ....

    public String imprimir(){
        return "Nome: " + nome +
               "\nIdade: " + idade +
               "\nSexo: " + sexo +
               "Endereço: " + end.imprimir();
    }
}
```

```
public class Endereco{
    //atributos
    private String logradouro;
    private String complemento;
    private int numero;
    private String cep;
    ....

    public String imprimir(){
        return "Logradouro: " + logradouro +
               "\nComplemento: " + complemento +
               "\nNúmero: " + numero +
               "CEP: " + cep;
    }
}
```

emerson@paduan.pro.br

Atenção!!!

```
public class AppPessoa {
    public static void main(String[] args) {
        Pessoa objPessoa = new Pessoa();

        System.out.println(objPessoa.imprimir());
    }
}
```



emerson@paduan.pro.br

Resolvendo

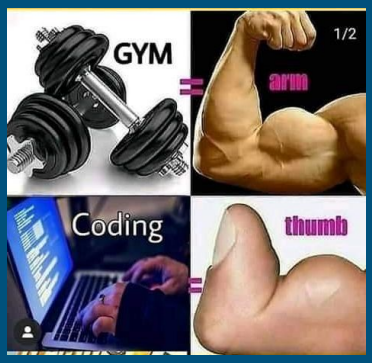
```
public class Pessoa{
    //atributos
    private String nome;
    private int idade;
    private char sexo;
    private Endereco end;

    //construtor default
    public Pessoa() {
        this.end = new Endereco();
    }

    public String imprimir(){
        return "Nome: " + nome +
               "\nIdade: " + idade +
               "\nSexo: " + sexo +
               "Endereço: " + end.imprimir();
    }
}
```


Exercício 6-4

1. Criar um relacionamento de Associação entre as classes Animal e Proprietário:
   Animal "Tem UM" *Proprietario*

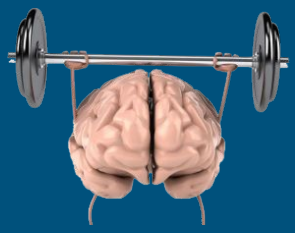
2. Criar a classe *Animal* com os seguintes atributos: nome, raça, cor, ano de nascimento e proprietário
3. Criar a Classe *Proprietario* com os seguintes atributos: nome e telefone
4. Criar uma classe para *Gerenciar* objetos da classe *Animal*
5. Criar a classe *AppAnimalProprietario* para:
   - Cadastrar Animal e Proprietário
   - Listar todos os animais cadastrados
   - Dada uma raça, listar o nome de todos os proprietários que tenham animal dessa raça

Extra



emerson@paduan.pro.br

Exercício 6-5



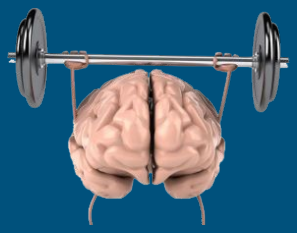
1. Criar a classe Livro com os atributos: título, autor, ISBN, gênero e preço.

2. Criar uma classe Livraria para gerenciar um ArrayList<> de Livro, com os seguintes métodos:
   – adicionar() : recebe um Livro e adiciona na lista
   – listar() : retorna uma String com os dados de todos os Livros cadastrados
   – pesquisarLivro() : recebe uma String como parâmetro que é o título do livro. Retornar os dados do livro.
   – pesquisarGenero() : recebe uma String como parâmetro que é o gênero do livro. Retornar o total de livros do gênero informado.
   – remover() : remove da lista um livro pelo título informado como parâmetro
   – calcularTotalAcervo() : retorna o valor total do acervo da Livraria

3. Criar a classe AppLivraria (main) que utiliza a classe Livraria

emerson@paduan.pro.br

Exercício 6-6



1. Criar uma classe Conta, que possua um número, um saldo e os métodos para obter dados da conta, depositar e sacar.
2. Crie as subclasses da classe Conta: ContaCorrente, ContaEspecial e ContaPoupanca.
   A ContaCorrente permite fazer saques somente se houver saldo suficiente. A ContaEspecial possui um limite que permite fazer saques se o saldo mais o limite da conta cobrir o valor pretendido de saque. A ContaPoupança faz saque se houver saldo, mas há uma taxa por operação.
   Além disso, a ContaCorrente deve reescrever o método deposita, com o objetivo de retirar uma taxa bancária de dez centavos de cada depósito.

3. Crie a classe GerenciaConta que permite adicionar, remover, listar, e fazer os saques e depósitos.

4. Crie uma classe AppContas com o método main contendo um menu com opções para realizar operações nas contas a partir da classe GerenciaContas.

emerson@paduan.pro.br